Code Obfuscation

David Pichardie

Master recherche Université Rennes 1
December 2011
Module SDL
Lecturer: Thomas Jensen

Code Obfuscation [1]

1. Most of these slides are taken from C. Collberg, C. Thomborson, D. Low, *A Taxonomy of Obfuscating Transformations*, `http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow97a/index.html`

Motivations

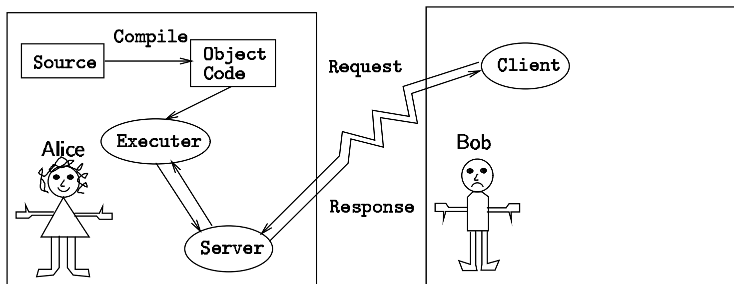
Software developers (Alice) want to protect their code from reverse engineering.

A rival developer (Bob) may steal proprietary algorithms or data structures.

Problem : it is becoming more common to distribute software in forms that are easy to decompile and reverse engineer.
Example : Java bytecode

Code obfuscation : Alice can protect her code by making reverse engineering so technically difficult that it becomes economically inviable.

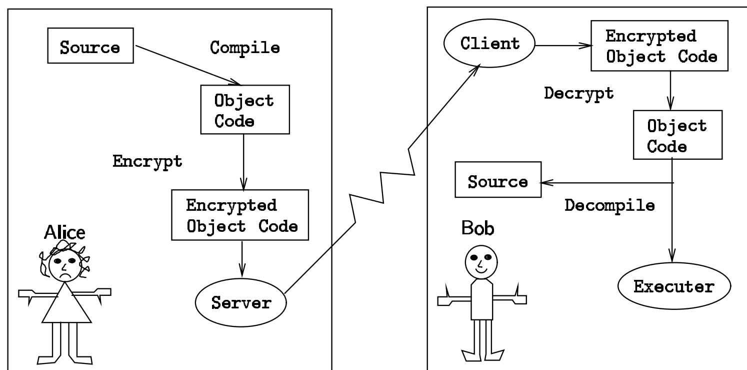
Other approach : Server-side protection



©C. Collberg

- ☺ Bob has not physical access to the code
- ☹ Bad performance

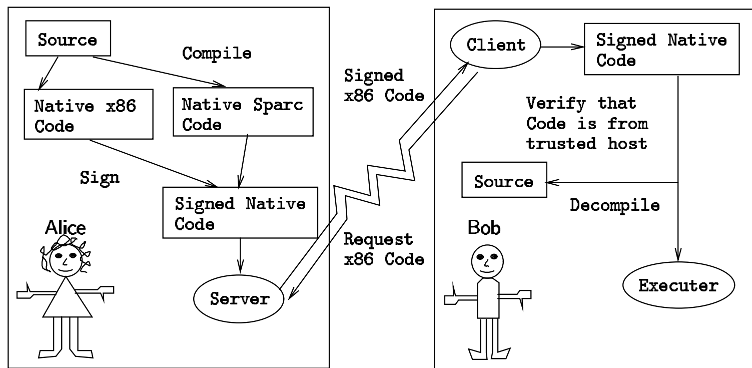
Other approach : protection by encryption



©C. Collberg

- ☺ Bob must decrypt the code to reverse engineer it
- ☹ decryption/execution must take place in hardware (≠ JVM)

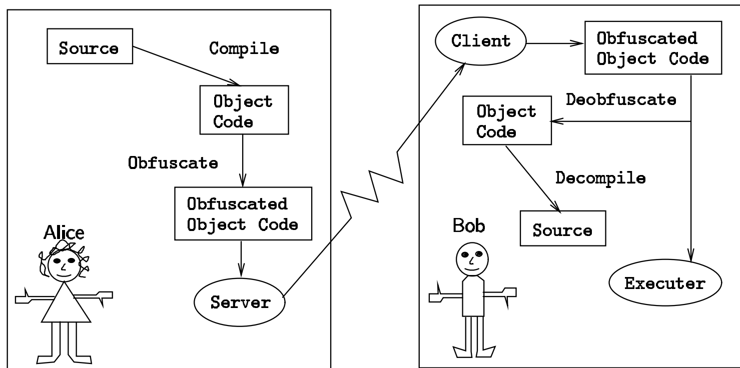
Other approach : protection by native code



©C. Collberg

- ☺ native code is more difficult to reverse engineer
- ☹ type safety is more difficult to ensure statically (unlike Java bytecodes)

Code obfuscation



©C. Collberg

- ☺ avoids previous shortcomings
- ☹ deobfuscation is always possible in theory (with enough time)

Obfuscation : definition

Let $P \underset{\mathcal{T}}{\rightarrow} P'$ be a transformation of a source program P into a target program P'. $P \underset{\mathcal{T}}{\rightarrow} P'$ is a *conservating transformation* if P and P' have the same *observable behavior* :

1. if P fails to terminate or terminates with an error, then P' may or may not terminate.
2. otherwise P' must terminate and produce the same output as P.

Quality

Collberg *et al.*[2] classify each transformations $\mathcal{T}$ according to several criteria

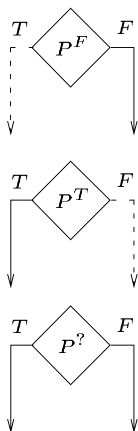
- **potency** how much obscurity $\mathcal{T}$ adds to the program
- **resilience** how difficult $\mathcal{T}$ is for a deobfuscator to undo,
- **stealth** how well code introduced by $\mathcal{T}$ fits in with the original code,
- **cost** how much computational overhead $\mathcal{T}$ adds to the obfuscated application.

2. C. Collberg, C. Thomborson, D. Low, *A Taxonomy of Obfuscating Transformations*

Trivial transformations

- remove the source code formatting informations
- remove comments (if any)
- scramble identifiers names

Opaque predicates



- The complexity of a program grows with the number of predicates it contains.
- A predicate P is opaque if its outcome is known at obfuscation time:

  $P^F \Rightarrow P$ is always `False`.

  $P^T \Rightarrow P$ is always `True`.

  $P^? \Rightarrow P$ is sometimes `True`, sometimes `False`.
- Obfuscating control transformations insert opaque predicates that are difficult for a de-obfuscator to evaluate.

©C. Collberg

Simple opaque constructs

Trivial opaque constructs can be cracked by a deobfuscator by a static local analysis :

$$
\begin{array}{l}
\{ \texttt{ int v, a=5; b=6;} \\
\quad \texttt{v}^{=11} \texttt{ = a + b;} \\
\quad \texttt{if } (\texttt{b} > \texttt{5})^{T} \cdots \\
\quad \texttt{if } (\texttt{random}(\texttt{1},\texttt{5}) < \texttt{0})^{F} \cdots \}
\end{array}
$$

©C. Collberg

Weak opaque constructs can be cracked by a deobfuscator by a static global analysis :

$$
\begin{array}{l}
\{ \texttt{ int v, a=5; b=6;} \\
\quad \texttt{if } (\cdots) \cdots \\
\qquad \vdots \; (\texttt{b is } \text{unchanged}) \\
\quad \texttt{if } (b < 7)^{T} \texttt{ a++;} \\
\quad \texttt{v}^{=36} \texttt{ = (a > 5)?v=b*b:v=b } \}
\end{array}
$$

©C. Collberg

Numeric opaque constructs

Number theory textbooks may give inspiration...

$\forall x, y \in \mathcal{I}, 7y^2 - 1 \neq x^2$
$\forall x \in \mathcal{I}, 2 \mid (x + x^2)$
$\forall x \in \mathcal{I}, 3 \mid (x^3 - x)$
$\forall n \in \mathcal{I}^+, x, y \in \mathcal{I}, (x - y) \mid (x^n - y^n)$
$\forall n \in \mathcal{I}^+, x, y \in \mathcal{I}, 2 \mid n \vee (x + y) \mid (x^n + y^n)$
$\forall n \in \mathcal{I}^+, x, y \in \mathcal{I}, 2 \not\mid n \vee (x + y) \mid (x^n - y^n)$
$\forall x \in \mathcal{I}^+, 9 \mid (10^x + 3 \cdot 4^{(x+2)} + 5)$
$\forall x \in \mathcal{I}, 3 \mid (7x - 5) \Rightarrow 9 \mid (28x^2 - 13x - 5)$
$\forall x \in \mathcal{I}, 5 \mid (2x - 1) \Rightarrow 25 \mid (14x^2 - 19x - 19)$
$\forall x, y, z \in \mathcal{I}, (2 \not\mid x \wedge 2 \not\mid y) \Rightarrow x^2 + y^2 \neq z^2$
$\forall x \in \mathcal{I}^+, 14 \mid (3 \cdot 7^{4x+2} + 5 \cdot 4^{2x-1} - 5)$

©C. Collberg

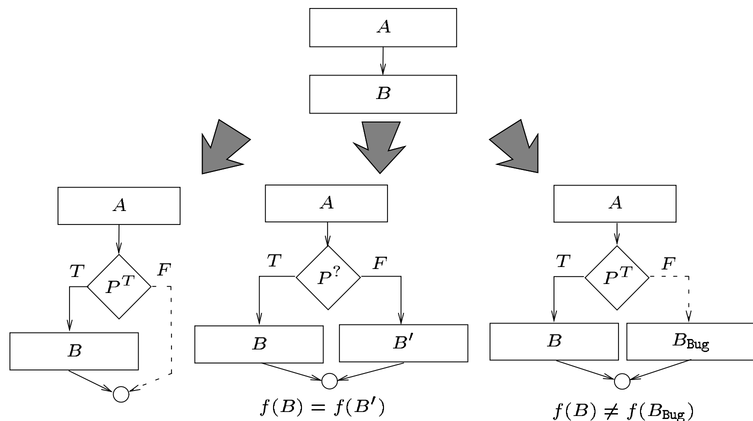
Opaque predicates : other techniques

The other techniques target the weaknesses of the static analyses

- Alias : the program maintains complex alias invariants such that some predicats on pointer equality are constant.
- Concurrency : thread interleaving is difficult to predict statically

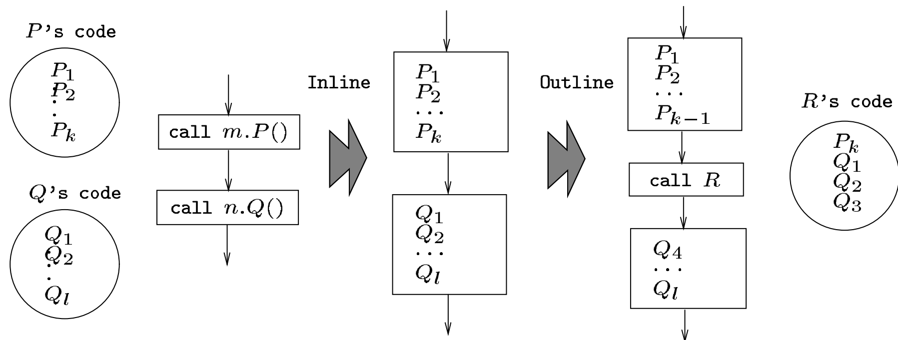
Using opaque predicates

Complexity (for the human understanding) increases with the number of predicates and nested levels of conditionals.
Application : insert Dead/Irrelevant code



©C. Collberg

Method inlining and outlining



©C. Collberg

Method interleaving

```
class C {
   A(T1 x){
      a1;a2;
   }
   B(T1 y;T2 z){
      b1;b2;
   }
}

{  C p=new C;
   p.A(x);
   p.B(y, z); }
```

$\overset{\mathcal{T}}{\Longrightarrow}$

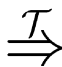
```
class C' {
   M(T1 xy;T2 y;int V){
      if (V == p) {a1;a2;}
      else        {b1;b2;}
   }
}

{  C' p=new C';
   p.M(x, c, V=p);
   p.M(y, c, V=q); }
```

©C. Collberg

Data transformations

Boolean variables and other variables of restricted range can be split into two or more variables.

```
(1)  bool A,B,C;
(2)  A = True;
(3)  B = False;
(4)  C = False;
(5)  C = A & B;
(6)  C = A & B;
(7)  C = A | B;
(8)  if (A) ···;
(9)  if (B) ···;
(10) if (C) ···;
```



```
(1')  short a1,a2,b1,b2,c1,c2;
(2')  a1=0; a2=1;
(3')  b1=0; b2=0;
(4')  c1=1; c2=1;
(5')  x=AND[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(6')  c1=(a1 ^ a2) & (b1 ^ b2); c2=0;
(7')  x=OR[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(8')  x=2*a1+a2; if ((x==1) || (x==2)) ···;
(9')  if (b1 ^ b2) ···;
(10') if (VAL[c1,c2]) ···;
```

©C. Collberg

String obfuscation

A constant string can be convert into a program that produces the string.

Exemple : `G(1)="AAA" G(2)="BAAAA" G(3)=G(5)="CCB" G(4)="XCB"`

```
String G (int n) {
  int i=0,k;
  String S;
  while (1) {
    L1:  if (n==1) {S[i++]="A";k=0;goto L6};
    L2:  if (n==2) {S[i++]="B";k=-2;goto L6};
    L3:  if (n==3) {S[i++]="C";goto L9};
    L4:  if (n==4) {S[i++]="X";goto L9};
    L5:  if (n==5) {S[i++]="C";goto L11};
        if (n>12) goto L1;
    L6:  if (k++<=2) {S[i++]="A";goto L6}
        else goto L8;
    L8:  return S;
    L9:  S[i++]="C"; goto L10;
    L10:  S[i++]="B"; goto L8;
    L11:  S[i++]="C"; goto L12;
    L12:  goto L10;
  }
}
```

©C. Collberg

Array transformations

Arrays can be

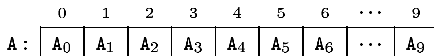
- split into several sub-arrays
- merged : two or more arrays are merged into one
- folded : increase the number of dimension
- flattened : decrease the number of dimension

Array splitting

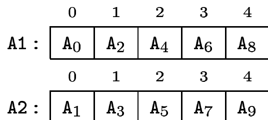
```
(1) int A[9];
(2) A[i] = ...;
```

$\Downarrow \mathcal{T}$

```
(1') int A1[4],A2[4];
(2') if ((i%2)==0) A1[i/2]=...
     else A2[⌊i/2⌋]=...;
```



$\Downarrow \mathcal{T}$



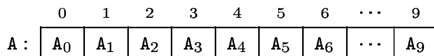
©C. Collberg

Array splitting

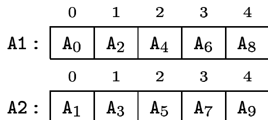
```
(1) int A[9];
(2) A[i] = ···;
```

$$\Downarrow \; \mathcal{T}$$

```
(1') int A1[4],A2[4];
(2') if ((i%2)==0) A1[i/2]=···
     else A2[⌊i/2⌋]=···;
```



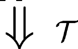
$$\Downarrow \; \mathcal{T}$$



©C. Collberg

Array merging

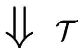
```
(3) int B[9],C[19];
(4) B[i] = ···;
(5) C[i] = ···;
```




```
(3') int BC[29];
(4') BC[3*i] = ···;
(5') BC[i/2*3+1+i%2] = ···;
```

	0	1	2	3	4	5	6	···	9
B :	B_0	B_1	B_2	B_3	B_4	B_5	B_6	···	B_9

	0	1	2	3	4	5	6	···	19
C :	C_0	C_1	C_2	C_3	C_4	C_5	C_6	···	C_{19}



	0	1	2	3	4	5	6	···	29
BC :	B_0	C_0	C_1	B_1	C_2	C_3	B_2	···	C_{19}

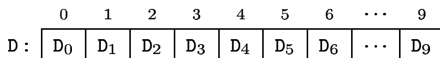
©C. Collberg

Array folding

```
(6) int D[9];
(7) for(i=0;i<=8;i++)
       D[i]=2*D[i+1];
```

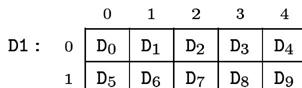
$\Downarrow \mathcal{T}$

```
(6') int D1[1,4];
(7') for(j=0;j<=1;j++)
       for(k=0;k<=4;k++)
         if (k==4)
           D1[j,k]=2*D1[j+1,0];
         else
           D1[j,k]=2*D1[j,k+1];
```



	0	1	2	3	4	5	6	···	9
D :	D_0	D_1	D_2	D_3	D_4	D_5	D_6	···	D_9

$\Downarrow \mathcal{T}$



D1 :	0	1	2	3	4
0	D_0	D_1	D_2	D_3	D_4
1	D_5	D_6	D_7	D_8	D_9

©C. Collberg

Array flattening

```
(8) int E[2,2];
(9) for(i=0;i<=2;i++)
      for(j=0;i<=2;i++)
        swap(E[i,j], E[j,i]);
```

$\Downarrow \mathcal{T}$

```
(8') int E1[8];
(9') for(i=0;i<=8;i++)
       swap(E[i], E[3*(i%3)+i/3]);
```

E :

	0	1	2
0	$E_{0,0}$	$E_{0,1}$	$E_{0,2}$
1	$E_{1,0}$	$E_{1,1}$	$E_{1,2}$
2	$E_{2,0}$	$E_{2,1}$	$E_{2,2}$

$\Downarrow \mathcal{T}$

E1 :

0	1	2	3	4	···	8
$E_{0,0}$	$E_{0,1}$	$E_{0,2}$	$E_{1,0}$	$E_{1,1}$	···	$E_{2,2}$

©C. Collberg

Conclusions

Theoretical limits

- Deobfuscation is always possible with enough time,
  - but some obfuscating transformations can be applied in polynomial time and require worst-case exponential time to deobfuscate (*i.e.* opaque predicates by aliasing).
- Obfuscation is also used by attackers (Bob) : viruses are duplicated and obfuscated to foil anti-virus software.
  - so deobfuscators are also necessary to ensure security !
- Obfuscation has received relatively little attention by the research community
  - but software companies use it
    Example : gMail midlet